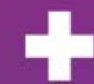
Информационная безопасность против Хайлоада

Рожков Денис
Руководитель разработки программного обеспечения в ООО «Газинформсервис»

Тарасов Георгий
Руководитель группы разработки СУБД
ООО «Газинформсервис»







О докладчиках



Рожков Денис

Руководитель разработки программного обеспечения компании «Газинформсервис».
В отрасли ИТ более 15 лет.
Участник крупных проектов заказчика федерального уровня.

Тарасов Георгий

Руководитель группы разработки СУБД компании «Газинформсервис».
Работает в области ИТ более 15 лет.



О компании «Газинформсервис»

Частная компания, основана в 2004 году в Санкт-Петербурге



9 филиалов
(в том числе 2 за рубежом)



Более **250** клиентов



Более **400** сертифицированных сотрудников



6 продуктовых линеек



Более **600** проектов в 31 субъекте федерации реализовано в 2021 году

- Разработка на базе проверенного ядра PostgreSQL
- Включена в реестр российского ПО
- Сертификация ФСТЭК - 5УД
- Профессиональная команда разработчиков ядра СУБД и решений на основе СУБД
- Многоуровневая техническая поддержка



Зачем?

- 152 ФЗ «О персональных данных» и вытекающие;
- Индустриальные требования и стандарты (SOX, PCI);
- Корпоративные риски.

Что?

- Идентификация и аутентификация;
- Управление доступом субъектов доступа к объектам доступа;
- Обеспечение целостности информационной системы и информации;
- Регистрация событий безопасности.



Что будем обсуждать?

СУБД

Реплики

Подключение

Подключение

Подключение

Аудит

Аудит

Log, Audit, Monitoring…
(SIEM, Zabbix и т.п.)

С самого начала у нас есть план, и мы будем его придерживаться!

1. Защита подключений
2. Защита на уровне данных
3. Реплики
4. Аудит

Цель: Как сочетаются требования высокой нагрузки и безопасности (хочу все быстро, надежно и безопасно).



Хотелось бы понять, а какое требование «важнее» для проекта в целом?

БЕЗОПАСНОСТЬ vs HIGHLOAD

0 : 0

Приступим…



Раунд 1. Защита подключений

1

СУБД

Реплики

Подключение

Подключение

Подключение

Аудит

Аудит

Log, Audit, Monitoring…
(SIEM, Zabbix и т.п.)



Сначала надо выяснить это:

- 1 бизнес-пользователь = 1 пользователь СУБД?
- Доступ к данным только через API?
- СУБД выделена в отдельный сетевой сегмент?
- Используется/нужен ли pooling/proxy и промежуточный слой?



Раунд 1. Защита подключений

POOLER

POOLER

СУБД

Реплики



Раунд 1. Защита подключений

POOLER POOLER

СУБД

Реплики

? ? ? ?



Уже потом можно применить это:

- Используйте решения класса database firewall.
- Требования к парольным политикам.
- IPsec (зачем этот монстр в цепочке доступа к кластерам СУБД?).
- Настраивайте SSL/TLS, если у вас нет сетевого разграничения СУБД.



Раунд 1. Защита подключений

POOLER POOLER

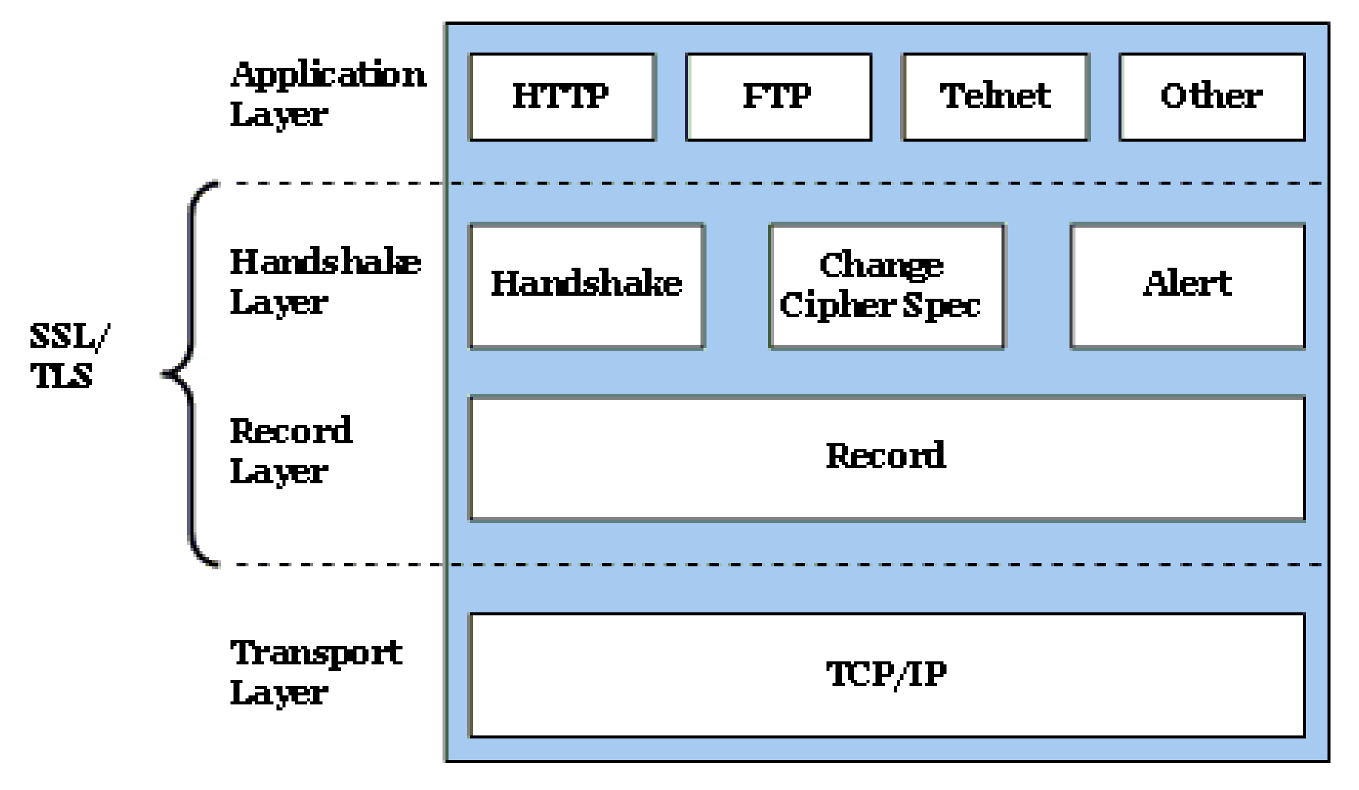
СУБД

Реплики

TLS? TLS? TLS? TLS?



Раунд 1. Защита подключений



Ref: https://sites.google.com/site/tlsssloverview/ssl-tls-protocol-layers

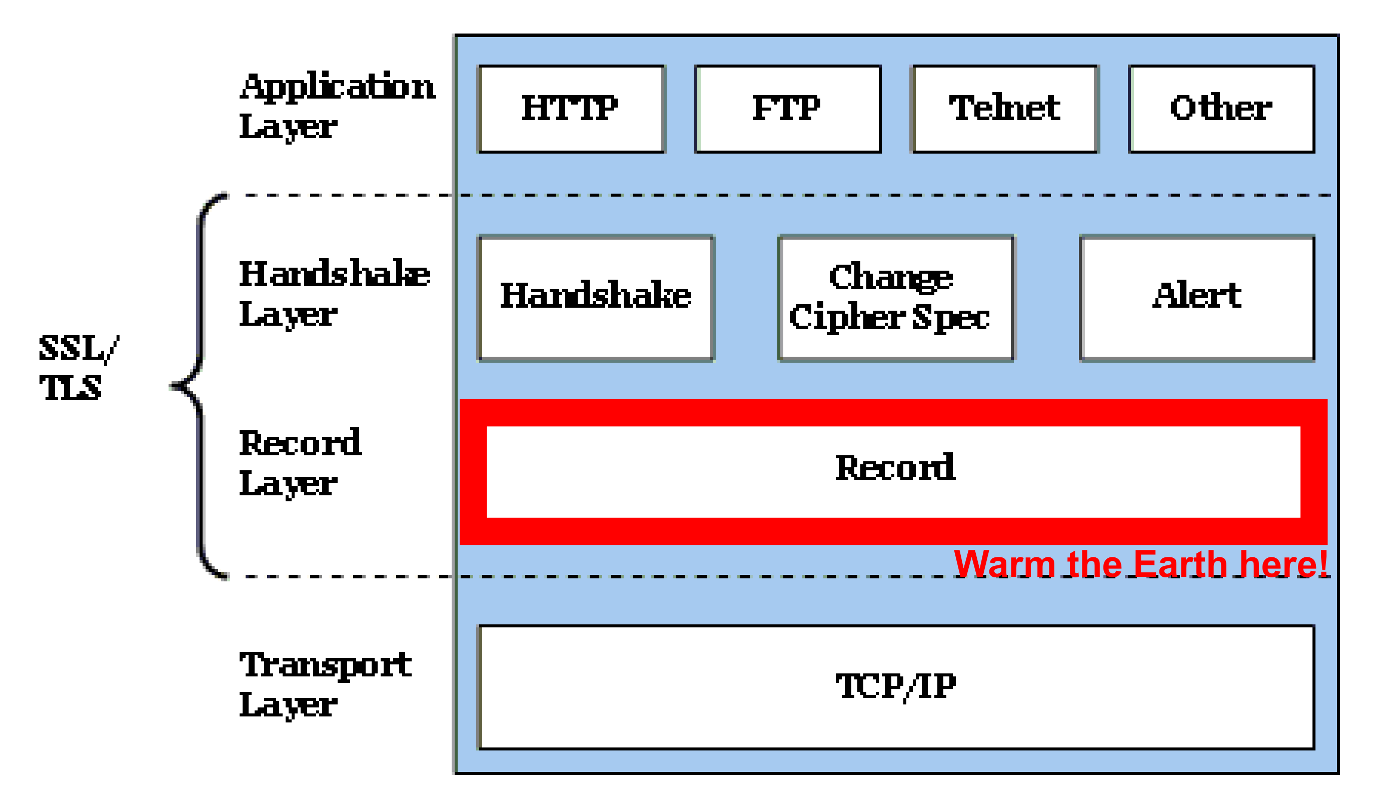
здесь грубо 50ms на соединение

здесь вся вычислительная нагрузка

здесь грубо 50ms на соединение



Раунд 1. Защита подключений



Ref: https://sites.google.com/site/tlsssloverview/ssl-tls-protocol-layers

здесь грубо 50ms на соединение

здесь вся вычислительная нагрузка

здесь грубо 50ms на соединение



Немного тестов без / с использованием SSL

Database: PosgtreSQL

Пробуем немного нагрузить, используя pgbench.

Устанавливается соединение при каждой транзакции:

pgbench.exe --connect -c 10 -t 5000 "host=192.168.220.129 dbname=taskdb user=postgres sslmode=verify-full sslrootcert=rootCA.crt sslcert=client.crt sslkey=client.key«

vs

pgbench.exe --connect -c 10 -t 5000 "host=192.168.220.129 dbname=taskdb user=postgres"



Немного тестов без / с использованием SSL

Database: PosgtreSQL

Пробуем немного нагрузить, используя pgbench.

Все транзакции выполняются в одно соединение:

```
pgbench.exe -c 10 -t 5000 "host=192.168.220.129 dbname=taskdb user=postgres sslmode=verify-full sslrootcert=rootCA.crt sslcert=client.crt sslkey=client.key"
```

vs

```
pgbench.exe -c 10 -t 5000 "host=192.168.220.129 dbname=taskdb user=postgres"
```



Немного тестов без / с использованием SSL

Database: PosgtreSQL

Пробуем немного нагрузить, используя pgbench.

Остальные настройки:

scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
number of transactions per client: 5000
number of transactions actually processed: 50000/50000



Результаты тестов

	NO SSL	SSL
Устанавливается соединение при каждой транзакции		
latency average	171.915 ms	187.695 ms
tps including connections establishing	58.168112	53.278062
tps excluding connections establishing	64.084546	58.725846
CPU	24%	28%
Все транзакции выполняются в одно соединение		
latency average	6.722 ms	6.942 ms
tps including connections establishing	1587.657278	1576.792883
tps excluding connections establishing	1588.380574	1577.694766
CPU	17%	21%



Что в итоге: *

- Нагрузка на ЦПУ (с увеличением числа коннектов > 10%)
- Увеличение таймингов (никогда часто не переоткрывать соединения, пулеры здесь в помощь)
- Уменьшение TPS (в общем незначительные проседание)
- Российский ГОСТ (явной разницы в потере произв. не зафиксировано)



Что в итоге:

*

- Нагрузка на ЦПУ (с увеличение числа коннектов > 10%)
- Увеличение таймингов (никогда часто не переоткрывать соединения, пулеры здесь в помощь)
- Уменьшение TPS (в общем не значительные проседание)
- Российский ГОСТ (явной разницы в потере произв. не зафиксировано)

БЕЗОПАСНОСТЬ vs HIGHLOAD

0 : 1



Раунд 2. Защита на уровне данных / доступа к данным

2

СУБД

Реплики

Подключение

Подключение

Подключение

Аудит

Аудит

Log, Audit, Monitoring…
(SIEM, Zabbix и т.п.)



Что можно применить?

- **На уровне данных**
  - Шифрование данных (полей)
  - Маскирование (преобразование при выводе / редакция) данных (masking)
- **На уровне доступа данных**
  - Разграничение доступа по ролям Security DBA \ Application DBA\ DBA
  - Ограничение доступа по строкам (RLS), когда GRANT/REVOKE уже недостаточно для вашей модели данных
  - Мандатный доступ, когда ролевая модель уже не справляется
- **Специальные методы**
  - Очистка памяти (требования ФСТЭК УД4)
  - Шифрование на уровне файлов/блоков (pg_cryogen / jacas)
  - End-to-end-шифрование (TDE + KMS в PostgreSQL)



Немного тестов с pgcrypto

Database: PosgtreSQL

Если надо скрыть отдельные поля

Модуль pgcrypto позволяет хранить в зашифрованном виде избранные поля.



Немного тестов с pgcrypto

Database: PosgtreSQL

Создадим таблицу с зашифрованными данными и с открытыми данными:

```
CREATE EXTENSION pgcrypto;

CREATE TABLE t1 (id integer, text1 text, text2 text);
CREATE TABLE t2 (id integer, text1 bytea, text2 bytea);

INSERT INTO t1 (id, text1, text2)
VALUES (generate_series(1,10000000), generate_series(1,10000000)::text, generate_series(1,10000000)::text);

INSERT INTO t2 (id, text1, text2) VALUES (
generate_series(1,10000000),
encrypt(cast(generate_series(1,10000000) AS text)::bytea, 'key'::bytea, 'bf'),
encrypt(cast(generate_series(1,10000000) AS text)::bytea, 'key'::bytea, 'bf'));
```



Немного тестов с pgcrypto

Выборка из таблицы без функции шифрования:

psql -c "\timing" -c "select * from t1 limit 1000;" "host=192.168.220.129 dbname=taskdb user=postgres sslmode=disable" > 1.txt

```
Секундомер включён.
 id | text1 | text2
------+-------+-------
 1 | 1 | 1
 2 | 2 | 2
 3 | 3 | 3
...
 997 | 997 | 997
 998 | 998 | 998
 999 | 999 | 999
 1000 | 1000 | 1000
(1000 строк)
```

Время: 1,386 мс



Немного тестов с pgcrypto

Выборка из таблицы с функцией шифрования:

```
psql -c "\timing" -c "select id, decrypt(text1, 'key'::bytea, 'bf'),
decrypt(text2, 'key'::bytea, 'bf') from t2 limit 1000;"
"host=192.168.220.129 dbname=taskdb user=postgres sslmode=disable" > 2.txt

Секундомер включён.
 id | decrypt | decrypt
------+------------+------------
 1 | \x31 | \x31
 2 | \x32 | \x32
 3 | \x33 | \x33
...
 999 | \x393939 | \x393939
 1000 | \x31303030 | \x31303030
(1000 строк)
```

Время: 50,203 мс



Почему шифрование не очень хорошо

Скорость

	без шифрования	Pgcrypto (decrypt)
Выборка 1000 строк	1,386 мс	50,203 мс
CPU	15%	35%
ОЗУ	40%	90%

Ключи могут быть перехвачены тем, кто имеет полный доступ к серверу баз данных, например, системным администратором.



Раунд 2. Защита на уровне данных / доступа к данным

- **Способы сокрытия данных, кроме шифрования:**
  - TABLESAMPLE (SELECT * FROM my_table TABLESAMPLE <SAMPLING METHOD>)
  - Замена / удаление (частичная замена) чувствительных данных
    - (UPDATE x SET y = 'КОНФЕДЕНЦИАЛЬНО')
    - (UPDATE x SET y = 'XXX-YYY-' || substr(y FROM … FOR …)
  - Генерирование случайных данных / шума
    - (UPDATE x SET y = random())
    - (UPDATE x SET y = y + 2.0 * random() / 3.0)
  - Перемешивание данных
  - Генерирование фиктивных данных

- **Все это**
  - **- или отнимает процессорное время (на «размаскирование»)**
  - **- или полезно для генерации тестовых наборов**



Раунд 2. Защита на уровне данных / доступа к данным

- **Ролевая / Избирательная / Мандатная модель управления доступом**

Проведем тест:

```
\copy (select 'create user u' || i || ';' from generate_series(1,500) i) to 'users.sql'
\i users.sql


\copy (
select 'create table t1('
union all
select string_agg('f' || i || ' integer', ',') from generate_series(1,500) as i
union all
select ');') to 'table.sql'
\i table1.sql
```



Раунд 2. Защита на уровне данных / доступа к данным

Разрешим каждому пользователю читать только «свое» поле

```
\copy (select 'grant select (f' || i || ') on table t1 to u' || i || ';' from generate_series(1,500) as i) to 'grant.sql'
\i grant1.sql
```

Разрешим пользователям запускать тестовую функцию

```
create or replace function func(x integer) returns integer as $body$
begin
  execute 'set role u' || x;
  execute 'select f' || x || ' from t1';
  return 0;
end;
$body$ language plpgsql;

grant execute on function func to public;
```



Раунд 2. Защита на уровне данных / доступа к данным

(I)

```
\set x random(1, 500)
begin;
select func(:x);
end;

pgbench.exe -c 10 -f bench1.sql -j 10 -n -t 20000 -U postgres postgres
```

(II)

```
begin;
select func(1);
end;

pgbench.exe -c 10 -f bench2.sql -j 10 -n -t 20000 -U postgres postgres
```



Раунд 2. Защита на уровне данных / доступа к данным

(I)

transaction type: bench1.sql
… skipped …
latency average = 0.714 ms
tps = 14013.996507 (including connections establishing)
tps = 14197.016750 (excluding connections establishing)

(II)

transaction type: bench2.sql
… skipped …
latency average = 0.692 ms
tps = 14459.490195 (including connections establishing)
tps = 14640.943760 (excluding connections establishing)



Раунд 2. Защита на уровне данных / доступа к данным

(I)

transaction type: bench1.sql
… skipped …
latency average = 0.714 ms
tps = 14013.996507 (including connections establishing)
tps = 14197.016750 (excluding connections establishing)

(II)

transaction type: bench2.sql
… skipped …
latency average = 0.692 ms
tps = 14459.490195 (including connections establishing)
tps = 14640.943760 (excluding connections establishing)

~ 3% overhead



Раунд 2. Защита на уровне данных / доступа к данным

Шифрование на уровне файлов данных

Разработано соответствующее расширение, шифрующее блоки симметричным шифром (ключ подставляется параметром через конфигурационный файл)

- задать ключ шифрования (256 бит):

```
SET jacas.key = "01234567890123456789012345678901";
```

- задать вектор инициализации (128 бит):

```
SET jacas.iv = "0123456789012345";
```

- создать зашифрованную таблицу:

```
CREATE TABLE table_a (g int) USING jacas;
```

tps = 1638.388343 (including connections establishing)

tps = 1510.705612 (including connections establishing)



Как БЕСПЛАТНО получить PostgreSQL with TDE

Качаем с сайта специальную версию с TDE

wget https://download.cybertec-postgresql.com/postgresql-12.X-TDE-1.0beta2.tar.gz



Как собрать Postgresql with TDE

Конфигурируем

```
./configure --prefix=/usr/local/pg12tde --with-openssl --with-perl \
 --with-python --with-ldap
```



Как собрать Postgresql with TDE

Компилируем

make install
cd contrib
make install



Как собрать Postgresql with TDE

Настраиваем управление ключами (KMS)

Надо ключ выводить в stdout

Вариант скрипта выдачи ключа при старте:

```
% cat /somewhere/provide_key.sh
#!/bin/sh
echo 882fb7c12e80280fd664c69d2d636913
```



Как собрать Postgresql with TDE

CREATING A DATABASE INSTANCE / CLUSTER

```
% initdb -D /some_path/db12tde -K /somewhere/provide_key.sh
 The files belonging to this database system will be owned by user "hs".
….
 Data page checksums are disabled.
 Data encryption is enabled.

 creating directory /some_path/db12tde ... ok
 creating subdirectories ... Ok
…
```

Как собрать Postgresql with TDE

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

pg_ctl -D /some_path/db12tde -l logfile start



Как запустить собранный Postgresql with TDE

```
% pg_ctl -D /some_path/db12tde start
 2020-01-29 11:54:19.131 CET [42193] LOG: starting PostgreSQL 12.3_TDE_1.0 on
 x86_64-apple-darwin19.2.0, compiled by Apple clang version 11.0.0 (clang-1100.0.33.17),
 64-bit
 2020-01-29 11:54:19.132 CET [42193] LOG: listening on IPv6 address "::1", port 5432
 2020-01-29 11:54:19.132 CET [42193] LOG: listening on IPv4 address "127.0.0.1", port
 5432
 2020-01-29 11:54:19.133 CET [42193] LOG: listening on Unix socket "/tmp/.s.PGSQL.5432"
 waiting for server to start....
 2020-01-29 11:54:19.151 CET [42197] LOG: database system was shut down at 2020-01-29
 11:54:05 CET
 2020-01-29 11:54:19.154 CET [42193] LOG: database system is ready to accept connections
 done
 server started
```



Как запустить собранный Postgresql with TDE

Помните скрипт для вывода ключа?

% grep encryption_key postgresql.conf

encryption_key_command = '/somewhere/provide_key.sh



А что сообщество думает?

Обсуждение с 2016 года
https://www.postgresql.org/message-id/flat/031401d3f41d%245c70ed90%241552c8b0%24%40lab.ntt.co.jpl

Five characteristics features:

1) Buffer-level data encryption and decryption
2) Per-table encryption
3) 2-tier encryption key management
4) Working with external key management services(KMS)
5) WAL encryption



А что сообщество думает?

commitfest.postgresql.org/28/2196/

Home / Commitfest 2020-07 / KMS - Internal key management system

KMS - Internal key management system

Edit | Comment/Review | Change Status

Title	KMS - Internal key management system
Topic	Server Features
Created	2019-07-02 05:25:54
Last modified	2020-12-10 06:48:41 (5 months ago)
Latest email	2020-09-08 08:23:23 (8 months, 1 week ago)
Status	2020-09: Returned with feedback 2020-07: Moved to next CF 2020-03: Moved to next CF 2020-01: Moved to next CF 2019-11: Moved to next CF 2019-09: Moved to next CF
Target version	
Authors	Masahiko Sawada (masahikosawada), Insung Moon (tsukiwamoon), Ibrar Ahmed (ibrar)
Reviewers	Cary Huang (cary)
Committer	
Links	Wiki
Emails	[Proposal] Table-level Transparent Data Encryption (TDE) and Key Management Service (KMS)



Что в итоге:

*

- Любое шифрование данных – нагрузка на CPU (отсюда overhead)
- Усложнение модели доступа данных => overhead
- Специальные методы доступа (data at rest) => overhead

БЕЗОПАСНОСТЬ vs HIGHLOAD

1 : 1



Раунд 3. Реплики

3

СУБД

Реплики

Подключение

Подключение

Подключение

Аудит

Аудит

Log, Audit, Monitoring…
(SIEM, Zabbix и т.п.)



В PostgreSQL:

- Синхронная / асинхронная реплика (согласованность данных)
- SSL / noSSL соединения между primary и replica (sslmode=ssl-verify)

Что получилось:

TPS	noSSL	SSL
SYNC	1562.525069	1558.958330
ASYNC	2083,868385	2049,406234



В PostgreSQL:

- Синхронная / асинхронная реплика (согласованность данных)
- SSL / noSSL соединения между primary и replica

БЕЗОПАСНОСТЬ vs HIGHLOAD

0 : 1



Раунд 4. Аудит

СУБД

Реплики

Подключение

Подключение

Подключение

Аудит

Аудит

4

Log, Audit, Monitoring…
(SIEM, Zabbix и т.п.)



Пишем логи и пытаемся потом там ловить.

Где и какой аудит можно включить?

- Коммерческие СУБД уровня Enterprise – там всё ОК.
- PosgreSQL – extension : pgaudit;
- PosgreSQL – default log (про это подробнее дальше).



Немного тестов с включением аудита запросов
Включено логирование всех запросов БД по всем параметрам

postgresql.conf

log_destination = 'stderr'
logging_collector = on
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 10MB
log_statement = **all**
log_min_duration_statement = 0
debug_print_parse = on
debug_print_rewritten = on
debug_print_plan = on
debug_pretty_print = on
log_checkpoints = on
log_connections = on
log_disconnections = on
log_duration = on
log_hostname = on
log_lock_waits = on
log_replication_commands = on
log_temp_files = 0
log_timezone = 'Europe/Moscow'



Немного тестов с включением полного аудита

Сервер: 1CPU 2,8ГГц, 2Гб ОЗУ, 40Гб HDD.
Database: PosgtreSQL

Включено максимальное логирование БД по всем параметрам.

Пробуем немного нагрузить, используя команды:

pgbench -p 5432 -U postgres -i -s 150 benchmark

pgbench -p 5432 -U postgres -c 50 -j 2 -P 60 -T 600 benchmark

pgbench -p 5432 -U postgres -c 150 -j 2 -P 60 -T 600 benchmark



Почему полный аудит не очень хорошо

	Без логирования	С логированием
Итоговое время наполнения БД	43,74 сек	53,23 сек
ОЗУ	24%	40%
CPU	72%	91%
Тест 1 (50 коннектов)		
Кол-во транзакций за 10 мин	74169	32445
Транзакций/сек	123	54
Средняя задержка	405 мс	925 мс
Тест 2 (150 коннектов при 100 возможных)		
Кол-во транзакций за 10 мин	81727	31429
Транзакций/сек	136	52
Средняя задержка	550 мс	1432 мс
Про размеры		
Размер БД	2251 МБ	2262 МБ
Размер логов БД	0 Мб	4587 Мб



А в жизни корпораций всё ещё хуже

- Данных много.
- Аудит надо не только через syslog в SIEM, но и в файлы тоже (ну так, на всякий случай).
- Для аудита нужна отдельная полка, чтоб не просесть по I\O дисков.
- Сотрудники ИБ: «Нам надо всё и да, ещё ГОСТ везде!».



*

БЕЗОПАСНОСТЬ vs HIGHLOAD

1 : 1

Что делать, если нет нужной фичи в СУБД?

“Пилите, Шура, пилите!”

Общий счет!

БЕЗОПАСНОСТЬ vs HIGHLOAD

0 : 1 1 : 1 0 : 1 1 : 1

2 : 4

Jatoba

Спасибо
за внимание!

+7 (812) 677-20-53
jatoba@gaz-is.ru

